A practical introduction to butterfly and adaptive resampling in Sequential Monte Carlo

Hakan Guldas * **A. Taylan Cemgil** * **Nick Whiteley** **
Kari Heine ***

* *Department of Computer Engineering Bogazici University, Istanbul, Turkey (e-mail: taylan.cemgil@boun.edu.tr, hakan.guldas@boun.edu.tr)*
** *School of Mathematics, University of Bristol, UK. (e-mail: nick.whiteley@bristol.ac.uk)*
*** *Department of Statistics, UCL, UK. (e-mail: k.heine@ucl.ac.uk)*

Abstract: Parallel and distributed computing technologies offer great potential for speed-up of Monte Carlo algorithms. However, in the development of most existing algorithms it has been implicitly assumed that implementation would be on a serial machine, so algorithm structure is often not well-suited to parallel architectures. In recent work the authors have studied the theoretical properties of sequential Monte Carlo algorithms involving a "butterfly" resampling method, whose conditional independence structure is intended to better match parallel and distributed architectures, with resampling broken down into stages, allowing sampling tasks for subsets of the particles to be handled concurrently. This paper provides a more practical overview of these methods, including consideration of adaptive resampling schemes, numerical results and an accessible account of theoretical properties.

Keywords: Particle filters, parallelization.

1. INTRODUCTION

Monte Carlo (MC) numerical methods are popular in a broad spectrum of applications across various applied fields, and Sequential Monte Carlo (SMC) methods in particular (Gordon et al., 1993; Doucet et al., 2001) are prominent due to their ease of implementation and wide applicability to inferential computation involving non-linear, non-Gaussian dynamical models. Historically, SMC methods, also known as particle filters, have been developed and analysed without much attention paid to the architecture of the computer systems on which they are implemented, the implicit assumption being that these algorithms would be deployed on a single processor system.

The modern picture is somewhat different: due to physical barriers and energy consumption, it is becoming increasingly difficult to design and develop processors with a faster central clock rate. Hence, the natural tendency for improved performance has been in moving towards parallel and distributed computation, realizing algorithms on systems consisting of slower but a very large number processing units. Many logical and physical arrangements for these very large number of processing units have been realized. However, traditional analysis of SMC methods has been focused on the behaviour of the error and does not provide any guidance for practical implementations on these modern architectures.

In modern parallel and distributed architectures, respecting data locality and avoiding unnecessary data movement is the key ingredient in the design of efficient algorithms. Here, consideration of the *communication pattern* - the structure via which computational elements exchange information, synchronize, balance computational load or access global memory - is crucial, as otherwise the potential benefits of parallel processing are easily lost (Lee et al., 2010; Suchard et al., 2010). The communication pattern of a SMC algorithm is naturally connected to the conditional independence structure of the stochastic process it simulates.

In this paper, we will focus on a new class of resampling algorithms called *butterfly resampling*, which the authors have recently proposed (Heine et al., 2014). The "butterfly" name reflects the fact that the conditional independence structure of the algorithm matches the pattern of the computation graphs of Cooley-Tukey Fast Fourier Transform, where each computation stage is called a butterfly. However, there several important practical and methodological issues which the theoretical work of Heine et al. (2014) does not address.

We present a practical overview of butterfly resampling algorithms with reference to their implementations on graphical processing units (GPUs). We also discuss their adaptive implementation based on monitoring the effective sample size. The performance of a practical implementation on a given system such as a such as a cluster, multicore processor or GPU's will depend on the details of a careful implementation. We provide simulation results on a GPU which are indicative of some of the key computation speed, stochastic and numerical error behaviours of but-

terfly resampling algorithms with comparison to standard resampling techniques.

2. RESAMPLING IN SMC

Resampling is a key operation in the design of stable SMC algorithms and its somewhat collective nature hinders the parallelization of the particle filters. There are various existing works on parallel implementations of SMC. Amongst several others Brun et al. (2002) considered particle filters in a distributed computing setting, Bolić et al. (2005) devised algorithms in which interaction occurs occasionally between blocks of particles, Hendeby et al. (2007) considered a GPU implementation, Vergé et al. (2013) have suggested algorithms with resampling on two hierarchical levels, and Murray et al. (2014) outlines some different approaches to parallel implementation of various existing resampling techniques and compares their efficiencies. Paige et al. (2014) propose an algorithm substantially different form of SMC method and, which involves a branching rather than resampling mechanism, so to deal with issues of synchronicity issues.

2.1 Augmented resampling

In this paper we present resampling in terms of a set of input particles $x_{\text{in}} = (x^i_{\text{in}})_{i=1}^N$, their weights $w_{\text{in}} = (w^i_{\text{in}})_{i=1}^N$, and the corresponding output quantities, $w_{\text{out}} = (w^i_{\text{out}})_{i=1}^N$, $w_{\text{out}} = (w^i_{\text{out}})_{i=1}^N$. From henceforth for any integer $n \geq 1$, we write $[n] := \{1, \ldots, n\}$.

Now let $m \geq 1$ and A_k, $k = 1, \ldots, m$ be non-negative matrices, each of size $N \times N$. We shall assume that

A1 each A_k is doubly stochastic,
A2 $(A_m A_{m-1} \cdots A_1)^{ij} = 1/N, \quad \forall i, j \in [N]$.

The starting point for butterfly resampling is the following *augmented resampling* procedure, which introduces m steps between the input and output.

– for $i \in [N]$, set $w^i_0 = w^i_{\text{in}}$, $\xi^i_0 = x^i_{\text{in}}$,
– for $k \in [m], i \in [N]$, set $w^i_k = \sum_j A^{ij}_k w^j_{k-1}$,

$$\text{and sample} \quad \xi^i_k \sim \frac{\sum_j A^{ij}_k w^j_{k-1} \delta_{\xi^j_{k-1}}}{\sum_j A^{ij}_k w^j_{k-1}},$$

– for $i \in [N]$, set $w^i_{\text{out}} = w^i_m$, $x^i_{\text{out}} = \xi^i_m$.

Our interest in this procedure is that the matrices A_k can be used to impose constraints on the conditional independence structure of random variables it involves. The specifics of these constraints are discussed a little later, in Section 3. However, introducing those details requires us to bring in another layer of notation, so before doing that we establish some important regularity and lack-of-bias properties of augmented resampling, in the sense of the following Lemma.

Lemma 1. For any $k =\in [m]$,

$$\max_i w^i_k \leq \max_i w^i_{\text{in}},$$

and for any integrable φ,

$$\mathbb{E}\left[\frac{1}{N}\sum_i w^i_k \varphi(\xi^i_k) \middle| x_{\text{in}}, w_{\text{in}}\right] = \frac{1}{N}\sum_i w^i_{\text{in}} \varphi(x^i_{\text{in}}), \quad (1)$$

and

$$\mathbb{E}\left[\frac{1}{N}\sum_i \varphi(x^i_{\text{out}}) \middle| x_{\text{in}}, w_{\text{in}}\right] = \frac{\sum_i w^i_{\text{in}} \varphi(x^i_{\text{in}})}{\sum_i w^i_{\text{in}}}.$$

Proof. The first claim follows from the row-stochasticity of the A_k and the fact that

$$w^i_k = \sum_j (A_k A_{k-1} \cdots A_1)^{ij} w^j_{\text{in}}.$$

For the second claim, using the column-stochasticity of each A_ℓ, or equivalently $\frac{1}{N}\sum_i A^{ij}_\ell = \frac{1}{N}$, we have

$$\begin{aligned}
&\frac{1}{N}\sum_i w^i_k \varphi(\xi^i_k) - \frac{1}{N}\sum_i w^i_{\text{in}} \varphi(x^i_{\text{in}}) \\
&= \sum_{\ell=1}^k \left\{ \frac{1}{N}\sum_{i=1}^N w^i_\ell \varphi(\xi^i_\ell) - \frac{1}{N}\sum_{i=1}^N \sum_{j=1}^N A^{ij}_\ell w^j_{\ell-1} \varphi(\xi^j_{\ell-1}) \right\} \\
&= \sum_{\ell=1}^k \frac{1}{N}\sum_{i=1}^N w^i_\ell \Delta^i_\ell,
\end{aligned}$$

where

$$\Delta^i_\ell = \varphi(\xi^i_\ell) - \frac{\sum_j A^{ij}_\ell w^j_{\ell-1} \varphi(\xi^j_{\ell-1})}{\sum_j A^{ij}_\ell w^j_{\ell-1}}.$$

From the "sample" part of the procedure, we have

$$\begin{aligned}
&\mathbb{E}\left[w^i_\ell \Delta^i_\ell \middle| x_{\text{in}}, w_{\text{in}} \right] \\
&= \mathbb{E}\left[w^i_\ell \, \mathbb{E}\left[\Delta^i_\ell \middle| x_{\text{in}}, w_{\text{in}}, (\xi^i_{\ell-1})_{i\in[N]} \right] \middle| x_{\text{in}}, w_{\text{in}} \right] \\
&= 0,
\end{aligned}$$

hence (1) holds.

For the third claim, note that property **A2** of the matrices implies that for all $i \in [N]$,

$$w^i_m = \sum_j (A_m A_{m-1} \cdots A_1)^{ij} w^j_{\text{in}} = \frac{1}{N}\sum_j w^j_{\text{in}}, \quad (2)$$

then re-arrange (1) with $k = m$. □

2.2 Adaptive strategies

In standard SMC algorithms, one may choose to perform resampling at only certain time steps, determined as the algorithm runs by monitoring the Effective Sample Size (ESS) and triggering resampling when the ESS falls below some chosen threshold, see (Whiteley et al., 2014) and references therein for background information and analysis of this approach. One can employ a similar strategy *within* the augmented resampling procedure outlined in the previous section. For instance with

$$\mathcal{E}^N_k = \frac{\left(\frac{1}{N}\sum_i w^i_k\right)^2}{\frac{1}{N}\sum_i (w^i_k)^2},$$

and $\tau \in (0, 1]$ a constant, define $k^\star := \min\{0 \leq k \leq m : \mathcal{E}^N_k \geq \tau\}$. Note that as per (2), w^i_m is in fact constant across i, so $\mathcal{E}^N_m = 1$ always. Thus $k^\star$ is well defined.

An alternative to the procedure of the previous section is then:

– for $i \in [N]$, set $w_0^i = w_{\text{in}}^i$, $\xi_0^i = x_{\text{in}}^i$,

– for $k \in [k^\star], i \in [N]$, set $w_k^i = \sum_j A_k^{ij} w_{k-1}^j$,

and sample $\xi_k^i \sim \dfrac{\sum_j A_k^{ij} w_{k-1}^j \delta_{\xi_{k-1}^j}}{\sum_j A_k^{ij} w_{k-1}^j}$,

– for $i \in [N]$, set $w_{\text{out}}^i = w_{k^\star}^i$, $x_{\text{out}}^i = \xi_{k^\star}^i$.

Using **A1**,

$$\frac{1}{N}\sum_i w_k^i = \frac{1}{N}\sum_i \sum_j (A_k \cdots A_1)^{ij} w_{\text{in}}^j = \frac{1}{N}\sum_i w_{\text{in}}^i,$$

and elementary manipulations then show that the value of $k^\star$ is determined entirely by τ, w_{in} and $A_1, \ldots, A_m$. Then it can be shown using very similar arguments to those in the proof of Lemma 1 (the details are left to the reader), that this adaptive procedure has the following lack-of-bias property:

$$\mathbb{E}\left[\left.\frac{\sum_i w_{\text{out}}^i \varphi(x_{\text{out}}^i)}{\sum_i w_{\text{out}}^i}\right| x_{\text{in}}, w_{\text{in}}\right] = \frac{\sum_i w_{\text{in}}^i \varphi(x_{\text{in}}^i)}{\sum_i w_{\text{in}}^i}.$$

3. BUTTERFLY RESAMPLING

3.1 Definition and properties

Let $N = r_1 r_2 \cdots r_m$ be a factorization of N with $r_k \geq 2$ for all $k = 1, \ldots, m$ and then let $A_1, \ldots, A_m$ be the family of matrices defined for $k = 1, \ldots, m$ as

$$A_k = I_{r_m} \otimes \cdots \otimes I_{r_{k+1}} \otimes \mathbf{1}_{r_k} \otimes I_{r_{k-1}} \otimes \cdots \otimes I_{r_1}, \quad (3)$$

where $\otimes$ denotes Kronecker product and for any positive integer n, we write $\mathbf{1}_n$ for the $n \times n$ matrix which has $1/n$ as every entry.

It follows by elementary properties of the Kronecker product that: **A1** is satisfied, each A_k is symmetric, $A_k A_l = A_l A_k$ for any $1 \leq k < l \leq m$, $A_m \cdots A_1 = \mathbf{1}_N$ so **A2** is satisfied, and each A_k matrix has exactly r_k non-zero elements on each row for $k = 1, \ldots, m$.

It can also be shown using elementary properties of the Kronecker product that the nonzero entries of the matrices $A_k, k = 1, \ldots, m$ are be characterized by the following modular congruence relations:

$$A_k^{ij} > 0 \iff$$
$$\lfloor \tfrac{i-1}{r_1 \cdots r_k} \rfloor = \lfloor \tfrac{j-1}{r_1 \cdots r_k} \rfloor \text{ and}$$
$$(i-1) \bmod (r_1 \cdots r_{k-1}) = (j-1) \bmod (r_1 \cdots r_{k-1}).$$

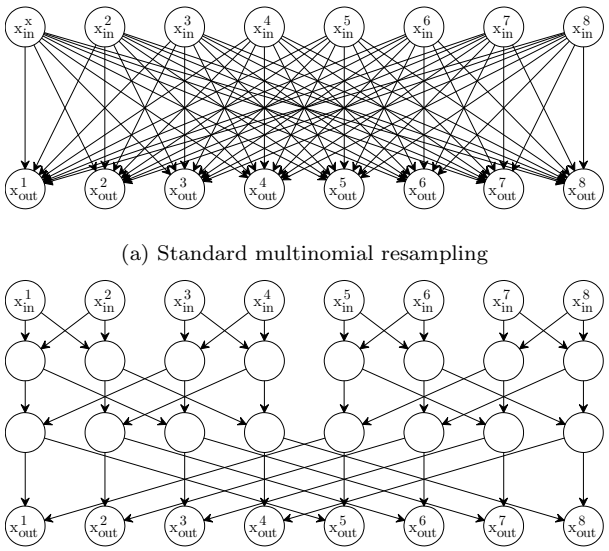
These relations are important because it is the zero entries of the matrices A_k which determine the conditional independence structure of the random variables in the augmented resampling scheme and influence which elements can be implemented algorithmically in parallel.

A specific instance of this setup is the case in which $r_1 = \cdots = r_m = r$, and the directed acyclic graph in Figure 1 shows the conditional independence structure of the resulting instance of augmented resampling when $N = 8$ and $r = 2$. The corresponding graph for multinomial resampling is also shown.

Heine et al. (2014) established central limit theorems for various instances of butterfly resampling. A particularly unusual feature of these results is that they show that butterfly resampling exhibits non-standard scaling: under mild regularity assumptions on x_{in}, w_{in}, the results of (Heine et al., 2014) can be applied in the case $r_1 = \cdots = r_m = r$ to show that the output from the procedure in Section 2.1 with the matrices as in (3) has the property that for any bounded test function φ,

$$\sqrt{N \log_r N}\left[\frac{1}{N}\sum_i \varphi(x_{\text{out}}^i) - \frac{\sum_i w_{\text{in}}^i \varphi(x_{\text{in}}^i)}{\sum_i w_{\text{in}}^i}\right] \Rightarrow \mathcal{N}(0, \sigma_r^2),$$

where the convergence is in distribution as N tends to infinity along the sequence $(r^m; m = 1, 2, \ldots)$. Thus *depending on the choice of* m *and* $r_1, \ldots, r_m$ butterfly resampling may converge more slowly than standard methods, which typically have $\sqrt{N}$ scaling as opposed to $\sqrt{N \log_r N}$. Thus speed-ups which butterfly resampling enjoys may trade off against loss in statistical performance.



(a) Standard multinomial resampling

(b) Butterfly resampling

Fig. 1. Interaction structures of resampling algorithms

3.2 Implementation issues

In order to frame some of our considerations when implementing butterfly resampling, it's convenient to first discuss some properties of the standard multinomial method, which one can obtain as a special case by setting $m = 1$ and $A_1 = \mathbf{1}_N$.

Pseudocode for multinomial resampling is given in algorithm 1 and involves parallel calls of the InversionSampling procedure, which itself consists of the generation of a uniform random number u in the interval $[0, W^N)$ and a binary search within the cumulative weights vector $W = (W^k = \sum_{i=1}^k w_{\text{in}}^i)_{k=1}^N$ to locate the subinterval $[W^{j-1}, W^j)$ such that $u \in [W^{j-1}, W^j)$. It returns the index j such that $u \in [W^{j-1}, W^j)$.

The pseudocode for butterfly resampling, which amounts to the augmented resampling procedure of Section 2.1 with the matrices (3), is given in algorithm 2, and expressed in terms of the following quantities

Algorithm 1 Multinomial Resampling on GPU

function MULTINOMIALRESAMPLE(w_{in}, x_{in})
  $W \leftarrow \text{PREFIXSUM}(w_{\text{in}})$
  for each $i \in \{1, \dots, N\}$ **do**
    $j^i \leftarrow \text{INVERSIONSAMPLING}(W, \{1, \dots, N\})$
    $x^i_{\text{out}} = x^{j^i}_{\text{in}}$
  end for
  return x_{out}
end function

(1) minimal elements of equivalence classes are given by:
$i_{r,k,p} = \lfloor \frac{p-1}{r_1 \cdots r_{k-1}} \rfloor r_1 \cdots r_k + (p-1) \bmod (r_1 \cdots r_{k-1}) + 1$,
(2) and index of equivalence class $[i]$ is given by:
$\mathcal{I}_{r,k,i} = \lfloor \frac{i-1}{r_1 \cdots r_k} \rfloor r_1 \cdots r_{k-1} + (i-1) \bmod (r_1 \cdots r_{k-1}) + 1$.

Algorithm 2 Butterfly Multinomial Resampling

function BUTTERFLYRESAMPLE(w_{in}, x_{in}, $r = (r_k)_{k=1}^m$)
  $w_0 \leftarrow w_{\text{in}}$ and $\xi_0 \leftarrow x_{\text{in}}$
  for $k \in \{1, \dots, m\}$ **do**
    for each $p \in \{1, \dots, N/r_k\}$ **do**
      $W_p \leftarrow \text{PREFIXSUM}(w_{k-1}^{[i_{r,k,p}]})$
    end for
    for each $i \in \{1, \dots, N\}$ **do**
      $j^i \leftarrow \text{INVERSIONSAMPLING}(W_{\mathcal{I}_{r,k,i}}, [i])$
      $\xi^i_k = \xi^{j^i}_{k-1}$ and $w^i_k \leftarrow W^{r_k}_{\mathcal{I}_{r,k,i}} / r_k$
    end for
  end for
  $w_{\text{out}} \leftarrow w_m, x_{\text{out}} \leftarrow \xi_m$
  return $w_{\text{out}}, x_{\text{out}}$
end function

The partitioning of the index set into equivalence classes correspond to division of the resampling operation into several resampling operations on smaller particle sets that can be carried out in parallel. On a GPU, these smaller resampling operations are mapped to thread blocks and within a thread block resampling is performed cooperatively by the threads. This allows inversion sampling to be carried out in shared memory and utilization of barrier synchronization and local memory for in-place propagation of particles.

4. EXPERIMENTAL RESULTS

We implemented the algorithms in CUDA C/C++ using CUDA Toolkit version 5.5. All simulations are run on a NVIDIA GeForce GTX680 GPU with CUDA capability 3.0.

4.1 Single Step of Resampling

We consider x_{in} samples a distribution π_0 and $w^i_{\text{in}} = g(x^i_{\text{in}})$, for some non-negative function g. For test function $\varphi(x) = x$ we consider estimation of $\hat{\pi}_0(\varphi) = \int (\varphi(x) g(x) / \pi_0(g)) \pi_0(dx)$ and measure performance in terms of numerical approximation of the MSE:

$$\mathbb{E}\left[\left(\sum_{i=1}^N \frac{w^i_{\text{out}}}{\sum_j w^j_{\text{out}}} \varphi(x^j_{\text{out}}) - \hat{\pi}_0(\varphi)\right)^2\right],$$

obtained over 1000 algorithm runs.

We consider two scenarios:

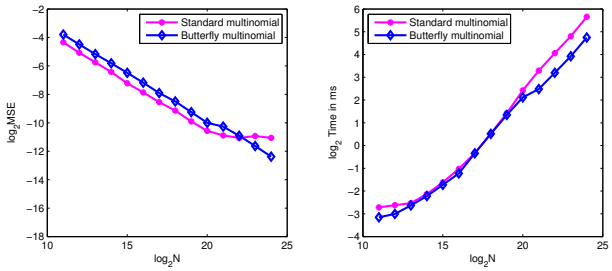
(1) π_0 is uniform distribution on $[-\sigma, \sigma]$ and $g(x) = \exp(-x^2/2\sigma^2)$,
(2) π_0 is Poisson distribution with parameter λ and $g(x) = \lambda^x / x!$,

with $\sigma = \lambda = 10$.

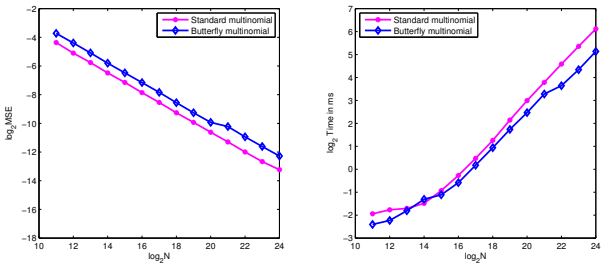
For each number of particles $N = 2^{11}, 2^{12}, \dots, 2^{24}$, we took $m = \lceil \log_R N \rceil$ where $R = 1024$ is the maximum number of threads per thread block and then selected the sequence $r_1, \dots, r_m$ obeying $N = r_1 \cdots r_m$ and giving the fastest performance.

We ran experiments in both single and double precision arithmetics. Results are plotted in figures 2 and 3. We see that butterfly resampling algorithm provides upto two times speed-up, since we utilized shared memory. However, with double precision, if look at the MSE against CPU time we do not see an improvement over standard multinomial resampling, this stems from the difference in the scaling factors of butterfly and standard multinomial resampling algorithms. While computation time grows at the order of $\log N$ for both algorithms, MSE of butterfly resampling decays at the order of $\log N/N$ and MSE of standard multinomial resampling decays at the order of $1/N$.

On the other hand, in single precision implementations, standard multinomial resampling exhibits numerically instability for large numbers of particles whilst butterfly resampling does not. We believe this is due the fact that in butterfly resampling, the prefix sum operation is applied to smaller subsets of the weights, which are in turn "smoothed out" over the stages of the algorithm.

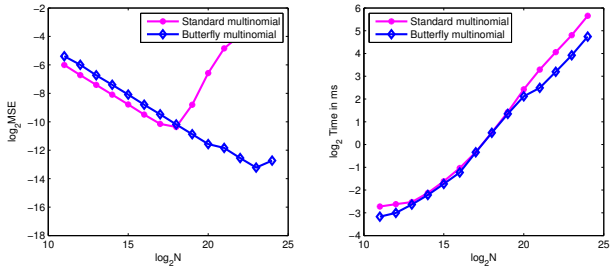


(a) Single precision.

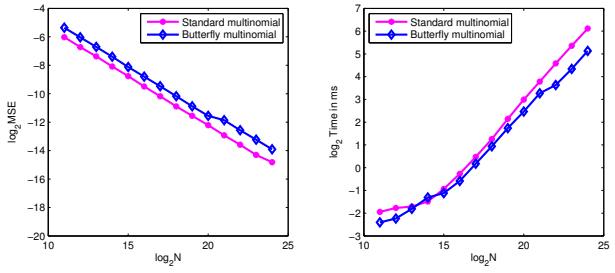


(b) Double precision.

Fig. 2. Comparison of standard and butterfly multinomial resampling, potential function $g(x) = \exp(-x^2/2\sigma^2)$.



(a) Single precision.



(b) Double precision.

Fig. 3. Comparison of standard and butterfly multinomial resampling, potential function $g(x) = \lambda^x/x!$.

4.2 Experiments with Particle Filters

In our particle filter experiments, we estimate filtering expectations $\hat{\pi}_t(\varphi)$ for $t \geq 1$ and for some test function φ. We measure the Monte Carlo errors by numerically estimating the the MSEs averaged over time (AMSE):

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\left(\hat{\pi}_t^N(\varphi) - \hat{\pi}_t(\varphi)\right)^2\right].$$

We implement two different particle filtering scenarios:

(1) resampling every time step, i) using multinomial and ii) butterfly resampling
(2) resampling adaptively, i) using multinomial resampling whenever the ESS of the weights falls below a given threshold, τ and ii) using the method of Section 2.2.

We use the following state space model for the filtering recursions:

$$x_{t+1} \sim f(x_t, .) = \mathcal{N}(x_t, \sigma_1),$$
$$g_t(x) = \{\exp((x - 0.5t)^2/2\sigma_2^2) + \exp((x + 0.5t)^2/2\sigma_2^2)\}/2.$$

This is a synthetic model that admits closed form computation of expectations of the function $\varphi(x) = x$ under a multimodal filtering distribution.

We use the same settings for m and $r_1, \ldots, r_m$ as in the previous experiments. Model parameters are set as $\sigma_1 = 0.1, \sigma_2 = 0.1$. We run each experiment for $M = 1000$ particle filter instances with $T = 100$ timestep and $\tau = 0.6$ for adaptive resampling, then compute AMSE for $\varphi(x) = x$.

MSE and speed results for bootstrap particle filters are shown in figure 4 and for adaptive resampling particle filters are shown in figure 5. As in single step resampling experiments, we see that butterfly resampling performs faster but produce more Monte Carlo error. In bootstrap PFs, butterfly resampling can provide upto 4 times speed-up but MSE per time does not differ substantially from standard multinomial resampling. However, in adaptive resampling PFs, butterfly resampling can provide upto 8 times speed-up and MSE difference is less than that of bootstrap case as a results of adaptivity, so we see an improvement over standard multinomial resampling in terms of MSE per time.

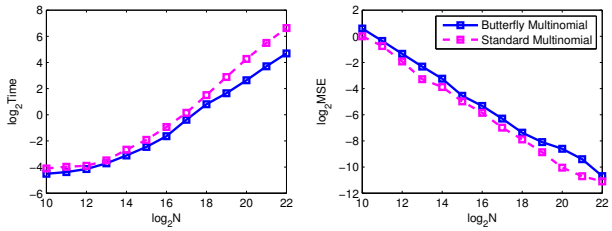


Fig. 4. Performance comparison of bootstrap particle filters.

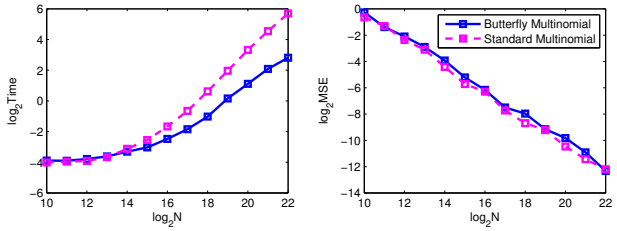


Fig. 5. Performance comparison of adaptive resampling particle filters.

4.3 Comparison of Resampling Strategies in a Practical Application

In our final experiment, we compare resampling strategies with constrained and full interactions within the context of a parameter estimation method that uses a particle filter as a subroutine. We have a family of state space models parameterized by some parameter vector $\theta \in \Theta$, where typically $\Theta = \mathbb{R}^d$, such that

$$X_0 \sim \mu_\theta,$$
$$X_n \mid \{X_{n-1} = x_{n-1}\} \sim f_\theta(x_{n-1}, \cdot),$$
$$Y_n \mid \{X_n = x_n\} \sim g_\theta(x_n, \cdot),$$

and a set of observations $y = (y_0, \ldots)$ and we perform maximum likelihood (ML) estimation of parameters θ to find the values of θ that maximize log-likelihood function defined as:

$$\ell(\theta) = \log\left(\int \mu_\theta(x_0)\prod_{i=1}^{n} f_\theta(x_{i-1}, x_i)\prod_{i=0}^{n} g_\theta(x_i, y_i)dx_{0:n}\right),$$

via sequential Monte Carlo expectation-maximization (SMCEM) algorithm (Olsson et al., 2008). SMCEM algorithm is based on expectation-maximization algorithm, a standard tool for ML estimation in the presence of intractable likelihoods, and uses particle filter as a subroutine.

In the experiments of this section, we follow example 4.2 of Olsson et al. (2008) and estimate parameters of the stochastic volatility model given as:

$$X_0 \sim \mathcal{N}(0, \sigma^2),$$
$$X_n \sim \mathcal{N}(\alpha X_{n-1}, \sigma^2), n \geq 1,$$
$$Y_n \sim \mathcal{N}(0, \beta^2 \exp(X_n)), n \geq 0.$$

We simulate $M = 10$ different observation sequences $(y^m_{0:n})_{m=1}^M$ of length $n = 100$ using the above model with parameters $\theta^*_1 = \alpha^* = 0.975, \theta^*_2 = \beta^* = 0.63$ and $\theta^*_3 = \sigma^* = 0.16$. For each observation sequence $y^m, m = 1, \ldots, M$ we perform SMCEM algorithm to obtain m parameter estimates $\hat{\theta}^m, m = 1, \ldots, M$ and compute MSE of the parameters given by the equations $\frac{1}{M}\sum_{m=1}^M (\hat{\theta}^m_i - \theta^*_i)^2, i = 1, 2, 3$.

We experiment with two different implementations of particle filter algorithms, one that uses standard multinomial resampling and another one that uses a special case of butterfly resampling algorithm. To simulate a situation with hard communication constraints and a fair comparison with standard multinomial resampling, we set the maximum radix sequence to 256 and we the following resampling strategy:

(1) for the particle filter with standard multinomial resampling, we perform at every $\lceil \log_{256} N \rceil$th timestep and defer resampling at other timesteps,
(2) for the particle filter with butterfly multinomial resampling, we perform one stage of butterfly resampling algorithm at every timestep by cyclically traversing the radix sequence.

For the choice of radix sequence we follow the guidelines we described in section 4.1.

We repeat this experiment for the particle numbers $\log N = 11, 13, \ldots, 21$ and plot the running time as a function of $\log N$ in figure 6. The accuracy of the parameter estimates, as measured by MSE, were omitted as they were qualitatively very similar. We observe that resampling under constrained interactions can provide upto three times speed-up over standard multinomial resampling while resulting effectively identical parameter estimates.

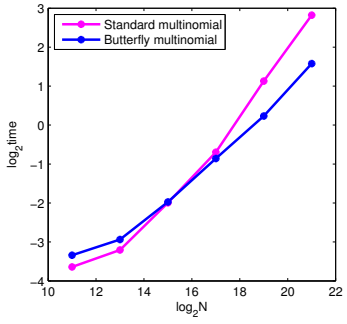


Fig. 6. Speed comparison of resampling algorithms in SMCEM algorithm.

5. CONCLUSIONS

Our goal was to investigate the tradeoff between particle interactions and estimation error, as well as how the reduced interaction structure can be used to obtain faster algorithms on standard hardware. We have focused on the GPU implementations of the resampling algorithms, and we compared the performance of butterfly resampling algorithm to standard multinomial resampling in terms of speed and Monte Carlo error. In our experiments with the reference implementations, we see that butterfly resampling can provide upto eight times speed-up in an adaptive resampling scenario with a competitive level of Monte Carlo error and upto three times speed-up in a parameter estimation problem with a competitive level of estimation quality.

We believe that the constrained interaction structure of the butterfly resampling provides additional flexibility in the design of resampling algorithms. We speculate that this additional flexibility may be exploited on alternative platforms such as distributed computer clusters or configurable hardware devices (e.g. field-programmable gate array – FPGA) to obtain faster algorithms without compromising estimation quality. A particularly interesting scenarioiu would be to use alternative sampling strategies, such as stratified and systematic sampling, within the butterfly method.

REFERENCES

Bolić, M., Djurić, P.M., and Hong, S. (2005). Resampling algorithms and architectures for distributed particle filters. *IEEE Trans. Signal Process.*, 53(7), 2442–2450.

Brun, O., Teuliere, V., and Garcia, J. (2002). Parallel particle filtering. *J. Parallel Distrib. Comput.*, 62(7).

Doucet, A., De Freitas, N., and Gordon, N. (eds.) (2001). *Sequential Monte Carlo methods in practice.* Springer, New York.

Gordon, N., Salmond, D., and Smith, A. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F*, 140(2), 107–113.

Heine, K., Whiteley, N., Cemgil, A., and Guldas, H. (2014). Butterfly resampling: asymptotics for particle filters with constrained interactions. ArXiv:1411.5876.

Hendeby, G., Hol, J., Karlsson, R., and Gustafsson, F. (2007). A graphics processing unit implementation of the particle filter. *15th European Signal Processing Conference (EUSIPCO).*

Lee, A., Yau, C., Giles, M.B., Doucet, A., and Holmes, C.C. (2010). On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. *J. Comput. Graph. Statist.*, 19(4), 769–789.

Murray, L.M., Lee, A., and Jacob, P.E. (2014). Parallel resampling in the particle filter. arXiv:1301.4019.

Olsson, J., Capp, O., Douc, R., and Moulines, r. (2008). Sequential monte carlo smoothing with application to parameter estimation in nonlinear state space models. *Bernoulli*, 14(1), 155–179.

Paige, B., Wood, F., Doucet, A., and Teh, Y.W. (2014). Asynchronous anytime sequential monte carlo. In *Advances in Neural Information Processing Systems*, 3410–3418.

Suchard, M.A., Wang, Q., Chan, C., Frelinger, J., Cron, A., and West, M. (2010). Understanding GPU programming for statistical computation: Studies in massively parallel massive mixtures. *J. Comput. Graph. Statist.*, 19(2), 419–438.

Vergé, C., Dubarry, C., Del Moral, P., and Moulines, E. (2013). On parallel implementation of Sequential Monte Carlo methods: the island particle model. *Stat. and Comput.*

Whiteley, N., Lee, A., and Heine, K. (2014). On the role of interaction in sequential Monte Carlo algorithms. *Bernoulli.* To appear.